Q4：寻找两个正序数组的中位数

1. 转换为找 **有序** 数组第 K 小的问题。 每次在上下数组找 k/2的位置，比较大小，小的数组左侧都可以被抛弃。

2. 然后找第 k-(j-start+1) 小即可。直到找第1小。即最小值即可。

```
 public double findMedianSortedArrays(int[] nums1, int[] nums2) {
         int n = nums1.length;
         int m = nums2.length;
         // 当 n + m 为 奇数时 left等于right
         // 假设 1，2，3，4 left就是2的位置，right就是 3的位置
         int left = (n + m + 1) / 2;
         int right = (n + m + 2) / 2;
         //将偶数和奇数的情况合并，如果是奇数，会求两次同样的 k 。
         // 找两个第k小的数即可。
         return (getKth(nums1, 0, n - 1, nums2, 0, m - 1, left) +
getKth(nums1, 0, n - 1, nums2, 0, m - 1, right)) * 0.5;
 }
 /*
 两个有序数组找第k小， 我们将两个数组并排，想办法找一条线，线左侧(包括线本身)有k个元素，那么线占用的两个数，就有可能是第K大，我们在上下两个数组，取 start_i+k/2 - 1 为线， 那么两个数组的线左侧都有 k/2-1个元素(不包含线)
 我们比较线上的两个值， 若up < down 那么up以及up左侧的都不可能是第k大。 因为，因为比up小的最多有 k/2-1 + k/2-1 = k-2个
 但是down数组左侧的元素，还可能大于up以及up右侧的元素。所以仍然可能是第k大，无法排除。
 这样我们就能排除一个数组的一侧。
 但后递归。
 */


 private int getKth(int[] nums1, int start1, int end1, int[] nums2, int
start2, int end2, int k) {
         int len1 = end1 - start1 + 1;
         int len2 = end2 - start2 + 1;
         //让 len1 的长度小于 len2，这样就能保证如果有数组空了，一定是 len1
         if (len1 > len2) return getKth(nums2, start2, end2, nums1, start1,
end1, k);
         if (len1 == 0) return nums2[start2 + k - 1];

         if (k == 1) return Math.min(nums1[start1], nums2[start2]);

         // 若 k/2已经超出边界，那么直接取最后一个。
         int i = start1 + Math.min(len1, k / 2) - 1;
         int j = start2 + Math.min(len2, k / 2) - 1;

         if (nums1[i] > nums2[j]) {
             return getKth(nums1, start1, end1, nums2, j + 1, end2, k - (j -
```

```
start2 + 1));
        }
        else {
            return getKth(nums1, i + 1, end1, nums2, start2, end2, k - (i -
start1 + 1));
        }
 }
```

Q15：三数之和 -两数之和可以用hashmap来解决。但是三数之和，就不行。

1. 采用先排序， 然后 基准线 + 双指针法。(三指针) i:a 在i右侧使用双指针，分别是b,c 若a > 0,则右侧一定找不到合适的bc.所以要求a<=0
2. 由于要去重，对于一对可行的bc使用后，要将left,right向内自动，直到b不是原来的b，c不是原来的c，或者left>right
3. 若对于 当前bc,a+b+c < 0 那么说明b小了， 因为更大的c已经使用过了，此时只能增加b.

Q42：接雨水

0. 对于 i < j 若 height[i] <= height[j] 那么i到j之间的水量可以根据 height[i]计算得到。
1. 对于.0, 无论i到j之间高度如何变化，只要对于任意k(i < k < j) height[k] < height[i]即可。不管中间是有单增，或者单减都可以计算出。

```
     *
*    *
*    *
*  * *
* ****
401215
则高度为4到5之间的水量都能计算到。
```

2. 根据.1.2, 若数组中间存在一个 最高的高度 maxh. 那么 位置0到maxh中间的水量都能计算到。高度为premaxh-height[i].
3. 计算过 premaxh 到 maxh之后，premaxh = maxh。

```
int maxleft = 0;
int ans = 0;
for(int i = 1; i < len; i++){
    if(height[i] >= height[maxleft]){
        int h = height[maxleft];
        for(int j = maxleft+1; j < i; j++){
            ans = ans + (h-height[j]);
        }
```

```
            maxleft = i;
        }
    }

    int maxright = len - 1;
    for(int i = len-2; i >= maxleft; i--){
        if(height[i] >= height[maxright]){
            int h = height[maxright];
            for(int j = maxright-1; j > i; j--){
                ans = ans +  (h-height[j]);
            }
            maxright = i;
        }
    }
    return ans;
```

4. 由于出现了 maxh. 那么maxh左侧的水量必定都已经计算完毕。 但是由于 maxh右侧没有 height[i] >= maxh所以都不会被计算到。

5. 计算len-1到 maxh之间的水量。方法同 左侧。

前缀和

Q134 : 加油站 是否可以绕一圈。从第 i 个加油站开往第 i+1 个加油站需要消耗汽油 cost[i] 升

```
gas  = [1,2,3,4,5]
cost = [3,4,5,1,2]
差数组:  -2 -2 -2 3 3
```

1. 我们先猜测前缀和最小位置的下一个位置是起始点。

2. 再猜测，只要整个数组是大于等于0就必然存在答案。

3. 首先是 最小前缀和的性质，假设最小前缀和的位置是 i, 那么一定存在j < i ,j到i的后一定小于0, 因为若其大于0，那么j才是最小前缀和。 而且由于i处是最小前缀和，那么i到k i < k，k是任意的i到end中间的一个数。 这个和 一定 > 0,否则i就不是最小。

4. 所以，当找到最小前缀和位置i, 那么从i+1,一定能走到end(因为3.后边i到任意K>i的和都是大于等于0的),

```
        public int canCompleteCircuit(int[] gas, int[] cost) {
                int totalsum = 0;
                int start = -1;
                int presum = 0;
                for(int i = 0; i < gas.length; i++){
                        totalsum += gas[i]-cost[i];

                        if(totalsum < presum){
```

```
                        start = i;
                        presum = totalsum;
                        }
                }
                if(totalsum < 0) return -1;
                return (start+1)%gas.length;
        }
```

Q135：分发糖果-每个孩子至少一个糖果，评分更高的孩子必须比他两侧的邻位孩子获得更多的糖果。求最少发糖数

1. 我们先考虑递增序列， 1234555， 只需要第一个发1个，然后后边到5发5个依次递增，第二个和第三个5只发1个。所以递增序列就是记住上一个发了pre，如果当前评分大于前一个那么就发pre+1, 如果等于就发1。然后更新pre。

2. 对于递减序列: 1，2， 7，6，5，4，3 通过递增计算知道，前三个发糖 1，2，3 1，2， 3 . 。 然后6发现递减了，于是对6发一颗糖。发糖数 k = 1 1，2， 3，1 5也是递减，那么就对 6和5都发一颗糖，发糖数 k = 2 1，2， 3，2，1 同理，遇到4，对654共发出 k=3课糖。 1，2 3，3，2，1 . 现在发现了错误， 7>6 那么7发的糖数应该大于6.所以发糖错误。

3. 当k = top处发糖的个数时，应该对top出也多发出一颗糖。也就是 k=3,不仅对654发，还要对7发。让k多做一次++即可。

```
 public int candy(int[] ratings) {
        int n = ratings.length;
        int ret = 1;
        int top = 1, k = 0, pre = 1;
        for (int i = 1; i < n; i++) {
        if (ratings[i] >= ratings[i - 1]) {
                k = 0;
                // 如果和前一个相等，这个位置就一定发一个。相当于数组从新开始的位
置。
                // 否则比前一个多发一个。
                top = ratings[i] == ratings[i - 1] ? 1 : pre + 1;
                ret += pre;
                //作为下一个的前一个。
                pre = top;
        } else {
                k++;
                if (k == top) {
                k++;
                }
                ret += k;
                //即当前位置发了一个。
                pre = 1;
        }
        }
        return ret;
 }
```

Q162 : 寻找峰值 假设 位置-1和len的大小是负无穷 。最简单的做法就是遍历一遍，第一个和最后一个特殊处理，即可。但是本题要求时间o(lgn)

```
while (l < r) {
    int mid = (l + r) / 2;
    if (nums[mid] > nums[mid + 1])
        r = mid;// mid位置就可能时山峰。在左侧
    else
        l = mid + 1; // mid+1的位置就可能是山峰，在右侧。
}
```

Q164 : 最大间距 : 给定一个无序的数组(非负)，找出数组在排序之后，相邻元素之间最大的差值。要求时间空间o(n)

1. 使用平均间距桶排序 我们可以肯定， 假设数组最大值最小值为 max，min, 那么每两个相邻的数之间的平均间距为 d = (max-min)/(n-1)
2. 那么由平均数原理，最大间距， dmax >= d 必成立。
3. 所以我们把x坐标，划分为 d大小的桶。 然后遍历每一个数，看其落在哪个桶内。而桶只保存桶内的最大最小值。
4. 为什么桶内其他值可以被抛弃，因为桶内的两个点的距离一定 < d . 所以桶内只存最大最小值。
5. 那么相邻两个桶的距离的最大值就是 ans.

Q189 : 旋转数组 空间要求 o(1)

1. 使用三次 reverse()函数即可。左侧一次，右侧一次，全局一次。

Q238 : 除自身以外数组的乘积 只能使用 ans一个数组和给出的num数组。不能使用其他额外空间。

1. 使用 ans数组先记录ans[i] = 前i-1个数的乘积。
2. 然后对ans从右往左算。ans[len-1]是正确的。 。使用R记录当前右侧所有数字的乘积。R = 1
3. 所以 ans[len-1] = ans[len-1] * R R = R*num[len-1] 依次类推。

窗口内删除最大值，后o(1)找第二大 双端队列，的单调递减栈：用于找第二大，递增栈用于找第二小

Q239 : 滑动窗口最大值 [1,3,-1,-3,5,3,6,7], k = 3 输出：[3,3,5,5,6,7]

1. 问题的关键是：若左端移出窗口，而他恰好是窗口内最大值。那么如何找到第二大的那个值。
2. 使用双端队列，的单调递减栈。若移出的元素不是队头那么直接删除即可，因为删除的不是窗口内最大值。

3. 若移出窗口元素是队头元素，说明删除元素是窗口内最大值。那么对头元素也要删除。然后以单调栈的方式把新元素加入队列尾部此时就可以确定新窗口的最大值，一定是队头。

Q287：寻找重复数 长度为n+1的数组,素为 1~n,则必定只有一个重复的数,找出这个重复的数

0. 要求不适用额外空间，不改变数组。

1. 桶归位。

```
设 num[0] = k
若k != num[k],则交换num[0] 和num[k]。这样元素k就归位了。
若k != num[k]则说明找到了重复。
但这样不好，原数组发生了改变.
```

2. **Q142**，等价于在 链表中，找循环部分开始的第一个结点。

```
        将数组看做是一个链表，则该链表必定包含环，而且不是所有的结点都在链表上，可能存在多个链表，但只有一个环链表。
        而且从num[0]一定能走到环上。

        对于数组 1 3 4 2 2    其实可以看作是一个链表。有5个结点。
        结点0 - > 结点1
        结点1 - > 结点3
        结点2 - > 结点4
        结点3 - > 结点2
        结点4 - > 结点2
        存在
        使用快慢指针来跑。 slow：fast：下标
        slow = nums[slow] : slow走向下一个结点，slow是下一个结点的下标。
        fast = nums[nums[fast]] :  一次走两个结点。

        int fast = 0;
        int slow = 0;
        slow = nums[slow];
        fast = nums[nums[fast]];
        while(fast != slow){
                slow = nums[slow];
                fast = nums[nums[fast]];
        }
        // 指向头结点
        fast = 0;
        while(fast != slow){
                slow = nums[slow];
                fast = nums[fast];
        }
        return fast;
```

Q324 : 摆动排序 II 重新排列成 nums[0] < nums[1] > nums[2] < nums[3]... 的顺序 **真hard**

1. 最容易想到的就是先排序，然后，从右半段逆序和前半段合并。

> 比如： 1,5,1,1,6,4 -> 1 1 1 4 5 6 -> 1 6 1 5 1 4
> 1,3,2,2,3,1 -> 1 1 2 2 3 3 -> 1 3 1 3 2 2 出现了错误。 本意是为了凸显先后的差
> 导致靠近数组中线两侧的 2 在合并时相遇。
> 如果 后半段不逆序。那么 就是 1 1 2 2 3 3合并。2和2就不会相遇
> 1 1 2 | 2 3 3 -> 1 2 1 3 2 3 ok

2. 针对1进行优化，首先为什么要排序？？。排序是为了划分数组，一侧小，一侧大，中间数位于中间。

> 首先求出中位数。中位数target两侧元素数量一样多。通过求第k小求出中位数。
> 然后按照三国国旗划分法，把小于target的集中到左侧，大于target集中到右侧，等于target的位于中间。
> 该算法也叫：three-way-partition算法。
> 然后化为左侧，和右侧。进行交替合并。左侧元素个数>=右侧元素
> 为什么前后不排序，也可以合并。只要后半段是大于target的即可。

```
int quickSelect_well(int[] arr, int k){
int start = 0; int end = arr.length-1;
while(true){
        int ind = partition(arr, start, end);
        if(ind+1 == k) return arr[ind];
        else if(ind+1 > k) end = ind-1;
        else start = ind+1;
}

int partition(int[] arr, int start, int end){
        int i = start; int j = end;
        int pivot = arr[start];
        while(i < j){
            if(i < j && arr[j] >= pivot) j--;
            arr[i] = arr[j];
            if(i < j && arr[i] <= pivot) i++;
            arr[j] = arr[i];
        }
        arr[i] = pivot;
        return i;
}

// 3-way-partition
void three_way_partition(int[] nums, int target){
    int i = 0, j = 0, k = nums.length - 1;
    while(j < k){
```

```
            if(nums[j] > target){
                //换到中间的数仍然可能大于或者小于target.
                swap(nums, j, k);
                --k;
            }else if(nums[j] < target){
                //换到中间的数一定是等于target的数。所以j可以移动。
                swap(nums, j, i);
                ++i;
                ++j;
            }else{
                // 相等则不动
                ++j;
            }
        }
    }
```

Q454 : 四数相加 II 四个等长数组， i,j,k,l .使得 A[i] + B[j] + C[k] + D[l] = 0

0. 总共有 n^4种组合。 可以将复杂度降到 n^2.
1. 在mapAB中存，每一个ai+bj 的数量。
2. 使用双层for，在map中找 -ck-dl 的数量count。 ans+= count.

Q480 : 滑动窗口中位数： 平衡树的root顶点恰好是中位数。但是实际的平衡树不好实现。**hard**

1. 因此使用 **双优先队列+延迟删除+平衡因子** 的做法
2. small：大顶堆 , 堆中元素是窗口内较小的一部分。
3. big : 小顶堆 , 堆中元素是窗口内较大的一部分。
4. balance : samll堆比big堆多出几个元素。 balance只能等于0，或者1. 表示两个堆元素相等，或者small比big堆多一个元素。此时表示是平衡的。
5. 首先把所有元素加入到 small堆(堆顶最大)。 然后从small堆中取 k/2个元素到 big堆中。这样在第一个窗口时，small保存的就是窗口内较小的一半，big保存的是最大的一半。 而且 small比big多 0个 或 1个元素。此时是平衡的。
6. 取中位数： 若k为奇数。 那么平衡时，小堆堆顶就是 中位数。 k为偶数时，两个堆顶的平均数就是 中位数。
7. 记账: 通过一个map: 记录要删除的元素key，和要删除key的数量value.
8. 滑动窗口： 先将 左侧删除(记账，并修改balance), 然后加入新元素(并修改balance)
9. 平衡： 滑动窗口后，balance可能的值 -2,-1,0,2 不可能是 1.0就不需要再平衡了。 对于-2，-1则要把big堆中元素丢入small堆中一个。 如果是2：则small堆中元素丢入big堆中一个。
10. 销账：只有堆顶元素在 账本中，才将堆顶删除。循环的。只要堆顶在账本中，就一直删除。

11. 取中位数： 销账之后才能取中位数。 然后重读 滑动窗口的步骤。

Q560 : 和为K的子数组： 和为k的 连续的 子数组 数组元素有负数

0. 前缀和 + 哈希表优化

1. 先得到前缀和数组。把前缀和 key:前缀和 value:key出现的次数 到map中

2. 遍历前缀和数组，现在就变成了 两数之和 问题。

3. 1，2两个过程可以合并在一个for循环中

4. 连续和问题 -> 前缀和问题 -> hash优化问题。

Q581 : 最短无序连续子数组 如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。 要求时间o(n)

0. [2,6,4,8,10,9,15] 对 [6, 4, 8, 10, 9] 进行升序排序 整个数组就会有序。输出： 子数组长度为 5

1. 特点： 子数组的最小值，一定大于等于子数组左侧的最小值。 子数组的最大值，一定小于等于 子数组右侧的最小值。

2. 方法1： 单调栈：

```
1. 数组从左到右入栈(下标)，维持单调递增栈。使用minind记录出栈元素的最小下标。
2. minind是什么？ 根据以上数组，2,6入栈，4会导致6出栈，这个6就是出栈最靠近左侧的位置。这个位置就是子数组的开始。
3. 如果后边有一个1，那么就会导致 2出栈，那么2的位置就是出栈最靠近 左侧的元素。
4. 同理从右到左入栈，记录 maxind
5. minind 到 maxind就是 子数组。
6. 时间 o(n),空间 o(n)
```

3. 方法2： 失序位置

```
1. 找左右第一个失序的位置。left,right.
2. 在 left, right之间找 minval最小值, maxval最大值。
3. 在0到left找看是否有 大于 minval的值，有的话，最靠近左侧的位置L就是 子数组的开始。
4. 同理 找右侧 最靠近右侧 的位置。小于 maxval 的位置R。
5. L 到 R 就是 子数组的边界。
```

Q697 : 数组的度 数组的度的定义是指数组里任一元素出现频数的最大值. 找到最短连续子数组的度和整体度相同。

0. 坑：当多个数字拥有最大度时，我们要取长度最小的那个(分布最集中)。

1. 使用map记录，开始位置，结束位置，该字符频率。空间o(n)

2. 特点：最短子数组中频率最高的元素x，x的频率就是子数组的度, 该子数组一定 以 x开始，并以x结尾。

3. 那么我们遍历整个数组，得到每个元素在数组中的开始位置start，和结束位置end，还有该元素的频率freq。

4. 所以这是一个 一对三 的映射关系。 num : (start,end,freq) 使用map<int,int[3]>存储即可。

5. 最终我们只对freq = k(整体数组的度) 的元素，求 min = Math.min(min, end-start+1)

Q992：K 个不同整数的子数组:若一个区间的数字总类恰好为 K, 那么就合法

0. 给定一个正整数数组 A，如果 A 的某个子数组中不同整数的个数恰好为 K，则称 A 的这个连续、不一定不同的子数组为好子数组。返回 A 中好子数组的数目。

1. K 个不同整数的子数组： 若一个区间的数字总类恰好为 K, 那么就合法，找出所有这样的子数组。

2. 子数组的子数组也可能合法。

```
[1,2,1,2,3]
[1,2], [2,1], [1,2], [2,3], [1,2,1], [2,1,2], [1,2,1,2].
```

3. 固定左边界时，移动右边界若合法res++, 那么可以找到 12,123,1212. 但是若这样，21就无法找到，因为right已经到了3.不能回溯。否则复杂度就不是o(n).

4. 现在，我们找数字种类最多为k的子数组数量。 那么对于A数组，他的最多为k的子数组数量n1, 最多为k-1的子数组数量n2.满足 res = n1-n2. 因为恰好为 k-1,k-2,....1的子数组是相同的。相减就恰好是 恰好种类为k的子数组数量。

5. res += right-left;的解释。假设本来是 ABC k=3, 然后变为 BCD,那么新的区间数就是 D,CD,BCD. 就是right-left。right新增加的一个位置，然后以这个位置结尾的区间数，就是新增的。

```
若有 12123
每当right后移动一个位置，就叠加新增的以right结尾的子数组。
1 right=0
2, 12 right = 1
1, 21, 121  right = 2
2, 12, 212, 1212, right = 3 然后right++
次数left要右移动，到  子数字 23  那么又有  3,23, 以3结尾。
这样就找到了 12123数组的所有数字种类数<=k的数量。
```

6. 这其实有集合的思想。 most(k)集合 - most(k-1) 就是剩余的 种类数恰好为k的集合。

Q1365：有多少个 小于当前数字 的数字 0 <= nums[i] <= 100

1.计数排序+前缀和思想 因为nums[i]<100，所以有了计数排序的可能，如果范围很大的话，就不能这么写了。

2. 先统计每个出出现的频率， 然后使用sum[i]保存 freq[0到i-1]的和。

3.sum就是答案。

JZ35 逆序对：本质就是求 元素左侧比元素大的数的个数。

1. 归并排序。

2. 归并阶段： 2 3 6 7 0 1 4 5

```
2 > 0 那么以0为右元素的逆序对就有 20 30 60 70
2 > 1 那么逆序对就有  21 31 61 71
2 < 4 4目前构不成  2去掉，数组状态： 3 6 7 - 4 5
同理3去掉， 合并 6 7 - 4 5  则有 64 74    剩余6 7 - 5
5： 65 75 然后 归并结束。
```

Q1423 : 可获得的最大点数：cardPoints = [1,2,3,4,5,6,1], k = 3。每次可以从头或者尾部拿一张。

1. 特点： 最终结果是，取走两侧一段，留下中间一段。

2. 要使得两侧和最大，那么就取长度为len-k的窗口，使得其和最小。

滑动窗口内第二大，第二小元素

Q1438 : 绝对差不超过限制的最长连续子数组

0. 窗口内 最大小 - 最小值 <= limit

1. 单调栈。使用两个双端队列的单调栈。一个单增，一个单减

```
1. 对right执行单调栈的入栈操作。

2. right入栈后，若当前窗口的最大值-最小值 >limit。 则left++一直收缩。

3. left++时，若left元素是队头元素，则把队头删除。因为删除的是最大值，或者最小值。
这样就暴露出第二大，或者第二小了。
```